Section 1

Question	Question title	Question type
1	Programming (Practice)	Matching
2	Expressions (Practice)	Text Entry
3	Conditions (Practice)	Code compile
4	Strings (Practice)	Text Entry
5	Loops (Practice)	Code compile
6	Methods (Practice)	Text Entry
7	Arrays (Practice)	Code compile
8	Classes (Practice)	Code compile
9	Inheritance (Practice)	Matching
10	Polymorphism (Practice)	Code compile
11	Exceptions (Practice)	Code compile

1 Programming (Practice)

Assume you want to convert a distance expressed in millimeters (given in the variable *int mm*) into precisely the same distance expressed in meters (calculated in the variable *double m*).

Which errors are present in the following Java statements, if any?

	Syntax Error	Runtime Error	Logic Error	No Error
double m = mm / 0;	○	○	○	○
double m = mm / 1000;	○	○	○	○
double m == mm / 1000;	○	○	○	○
double m = mm / 1000.0;	○	○	○	○

(1 point per correct answer; max. 4 points)

2 Expressions (Practice)

Given the following variable declarations:

```
boolean b = true;
char c = 'C';
double d = 16.18;
int i = 7;
String s = "Java";
```

What is the output of the following Java statements?

System.out.println(c -= i - 5); // prints _____

*System.out.println(d - i * 2);* // prints _____

System.out.println(d < i == !b); // prints _____

System.out.println((int) d % i); // prints _____

System.out.println(--i); // prints _____

System.out.println(d + s.length()); // prints _____

(1 point per correct answer; max. 6 points)

3 Conditions (Practice)

The following Java program reads the coordinates *(x1, y1)* and *(x2, y2)* of two diagonally opposed corners of a rectangle from the keyboard. (Note: These may be any corners, not necessarily first the bottom-left and then the top-right one.)

Complete the program so that it

- reads the coordinates of an arbitrary point *(x, y)* from the keyboard
- prints "in" if that point is inside the rectangle or on its boundary
  or "out" if it is outside the rectangle

Complete the program:

Test case #	Input	Expected output
1	-6 -4 10 12 2 -2	in
2	10 12 -6 -4 2 -2	in
3	10 12 -6 -4 -10 20	out
4	-6 -4 10 12 2 20	out
5	-6 -4 10 12 -6 12	in

```
import java.util.Scanner;

class Geometry {

    public static void main(String[] args) {
        Scanner stdin = new Scanner(System.in);
        int x1 = stdin.nextInt();
        int y1 = stdin.nextInt();
        int x2 = stdin.nextInt();
        int y2 = stdin.nextInt();

        // INSERT CODE HERE
    }

}
```

Test code

(Autograder results are indicative only -- implementation is graded manually; max. 10 points)

4 Strings (Practice)

Consider the following Java method:

```
public String encode(String name) {
    String output = "";

    int index = 0;
    do {
        output += name.charAt(index);
        index = name.indexOf(' ', index) + 1;
    } while (index != 0);

    index = name.lastIndexOf(' ') + 1;
    while (output.length() < 3) {
        output += name.charAt(++index);
    }

    return output;
}
```

What is the return value of the following method invocations?

encode("Charles Antony Richard Hoare"); // returns ____

encode("Alan Mathison Turing"); // returns ____

encode("Ada Lovelace"); // returns ____

encode("Al-Khwarizmi"); // returns ____

(1 point per correct answer, max. 4 points)

Hints: The above code uses the following methods of the *String* class:

- ***public char charAt(int index)*** returns the character at the specified *index* of this string (the first character is at index 0).
- ***public int indexOf(char ch, int fromIndex)*** returns the index of the first occurrence of the character *ch* in this string that is greater than or equal to *fromIndex*, or -1 if *ch* does not occur.
- ***public int lastIndexOf(char ch)*** returns the index of the last occurrence of the character *ch* in this string, or -1 if *ch* does not occur.
- ***public int length()*** returns the length of this string.

5 Loops (Practice)

Write a Java program that

- reads a String *s* from the keyboard
- loops through *s* to print its characters in reverse order

Complete the program:

Test case #	Input	Expected output
1	Hello	olleH

```
import java.util.Scanner;

class Reverse {
    public static void main(String[] args) {
        Scanner stdin = new Scanner(System.in);
        String s = stdin.nextLine();

        // INSERT CODE HERE

    }
}
```

Test code

(Autograder results are indicative only -- implementation is graded manually; max. 3 points)

6 Methods (Practice)

Consider the following Java methods:

```
public static void swap(int a, int b) {
    int c = a;
    a = b;
    b = c;
}

public static void swap(int[] a, int[] b) {
    int[] c = a;
    a = b;
    b = c;
}

public static void swap(int[] arr, int a, int b) {
    int c = arr[a];
    arr[a] = arr[b];
    arr[b] = c;
}
```

What is the output of the following program?

```
public static void main(String[] args) {
    int m = 4;
    int n = 2;
    swap(m, n);
    System.out.println(m + ", " + n);        // prints
```

```
    int[] p = {1, 2, 3};
    int[] q = {4, 5, 6};
    swap(p, q);
    System.out.println(p[0] + ", " + q[0]); // prints
```

```
    int[] r = {7, 8, 9};
    swap(r, 0, 1);
    System.out.println(r[0] + ", " + r[1]);   // prints
}
```

(1 point per correct answer, max. 3 points)

7 Arrays (Practice)

Write a Java program that

- reads a series of seven integer values from the keyboard into an array
- calculates the average of the array values
- prints the number (*count*) of values that are lower than the average

Complete the program:

Test case #	Input	Expected output
1	1 2 3 4 5 6 7	3
2	1 1 1 1 1 10 20	5

```
import java.util.Scanner;

class Counter {
    public static void main(String[] args) {
        Scanner stdin = new Scanner(System.in);
        int count = 0;
        // INSERT CODE HERE

        System.out.println(count);
    }
}
```

Test code

(Autograder results are indicative only -- algorithm is graded manually; max. 10 points)

8 Classes (Practice)

Implement a Java class with the necessary attributes and methods (constructor, getters and setters) and suitable visibilities to reflect the following scenario:

A person is given a kennitala upon birth. The kennitala can subsequently only be read but not be modified. The name of the person is initially empty, but can be modified and read anytime.

Hint: Use Strings for all attributes.

Implement the class:

Test case #	Input	Expected output
-		

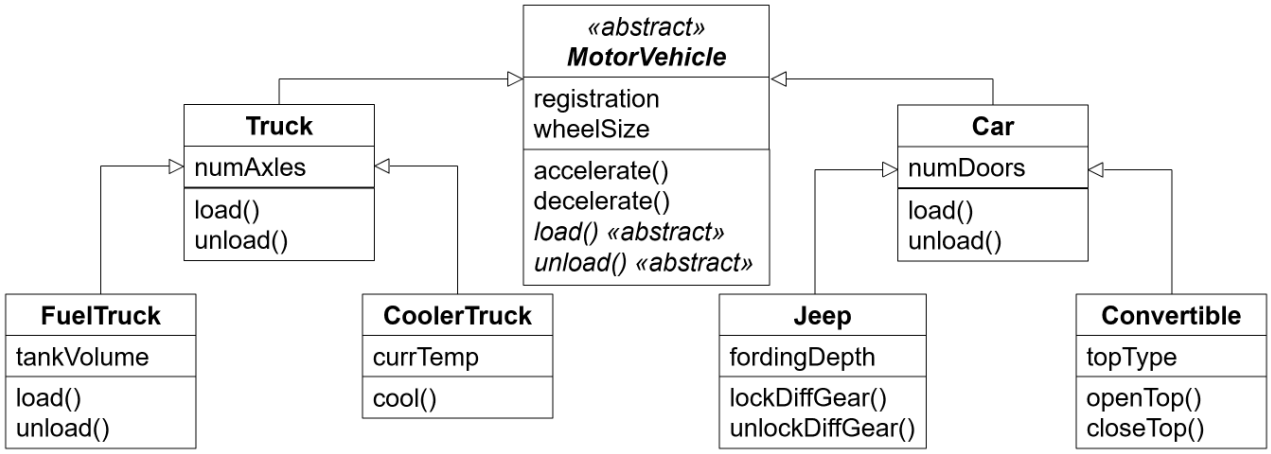
```
// INSERT CODE HERE

```

Test code

(Ignore autograder output -- implementation is graded manually; max. 10 points)

9 Inheritance (Practice)

Assume you have a collection of Java classes, implemented according to the following UML class diagram:



Would the following statements compile or not?

	compiles	error
Truck t = new FuelTruck(); t.numAxles = 3;	○	○
Jeep j = new Jeep(); j.lockDiffGear();	○	○
MotorVehicle mv = new Truck(); mv.unload();	○	○
Convertible conv = new Convertible(); conv.load();	○	○
CoolerTruck ct = new Truck(); ct.currTemp = -18;	○	○
MotorVehicle mv = new MotorVehicle(); mv.accelerate();	○	○
Car car = new Convertible(); car.openTop();	○	○

(1 point per correct answer; max. 7 points)

10 Polymorphism (Practice)

Complete the following Java classes that represent different types of bank accounts:

The abstract class *Account* represents a generic bank account, which is characterized by its identifying *number* and *balance* (i.e. the amount of money in the account). Its concrete subclasses *Checking* and *Savings* represent checking and savings accounts, respectively. Checking accounts are characterized by their permitted *overdraft* (i.e. the amount of money one can widthdraw in excess of the current balance), and savings accounts are characterized by their *interest* rate. A skeletal implementation of these classes is given below.

Complete the given implementation with the following elements, so that the *Test* class will compile and pass the provided test case:

- Add the above-mentioned **attributes** to the *Account*, *Checking* and *Savings* classes where it is most efficient.
- Add the following **methods** to the *Account*, *Checking* and/or *Savings* classes where it is most efficient, so that the *Test* class will compile:
  - ***public void deposit(int amount)*** adds the given *amount* to the account's balance.
  - ***public int available()*** returns the maximum amount available for widthdrawal (Hint: For savings accounts, this is the balance; while for checking accounts, it is the balance plus overdraft)
  - ***public void setOverdraft(int overdraft)*** sets a checking account's overdraft to the given value
- Add a *Currency* **subclass** of *Checking* with the minimum necessary implementation to reflect a foreign currency account, characterized by its *exchange* rate.

Complete the program:

Test case #	Input	Expected output
1		11000 1000 11000

```
abstract class Account {
    // INSERT ATTRIBUTE(S) HERE AS NEEDED

    public Account(int number) {
      this.number = number;
      balance = 0;
    }

    // INSERT METHOD(S) HERE AS NEEDED
}

class Checking extends Account {
    // INSERT ATTRIBUTE(S) HERE AS NEEDED

    public Checking(int number, int overdraft) {
        super(number);
        this.overdraft = overdraft;
    }

    // INSERT METHOD(S) HERE AS NEEDED
}

class Savings extends Account {
    // INSERT ATTRIBUTE(S) HERE AS NEEDED

    public Savings(int number, double interest) {
```

```
    public SavingsAccount(int number, double interest) {
        super(number);
        this.interest = interest;
    }

    // INSERT METHOD(S) HERE AS NEEDED
}
```

Test code

(Autograder results are indicative only -- implementation is graded manually; max. 15 points)

11 Exceptions (Practice)

The following Java program reads an integer from the keyboard and prints its squared value. However, if the user's input cannot be converted into an integer (e.g. "abc"), the program will terminate with an *InputMismatchException*.

Modify the program so that it discards the entered value and reads another integer if

- the entered value is larger than 100
- the entered value cannot be converted into an integer

(Hint: If a keyboard input could not be converted into an integer, you need to clear it by calling the Scanner's *nextLine()* method before you can attempt to read another integer value.)

Modify the program:

Test case #	Input	Expected output
1	5	25
2	150 5	25
3	abc 5	25
4	abc def 150 200 5	25

```
import java.util.Scanner;
import java.util.InputMismatchException;

class ExceptionHandling {
    public static void main(String[] args) {
        // MODIFY THIS METHOD
        Scanner stdin = new Scanner(System.in);
        int input;

        input = stdin.nextInt();

        System.out.println(input * input);
    }
}
```

Test code

(Autograder results are indicative only -- implementation is graded manually; max. 8 points)